Minimum cost flows

Amaury Pouly

November 23, 2010

Contents

1 Introduction **2**

1.1 Mathematical formulation . . . 2

1.2 Applications to graph theory . . . 2

1.2.1 Shortest path problem . . . 3

1.2.2 Maximum flow problem . . . 3

1.3 Applications to real world problems . . . 3

1.3.1 Determining an optimal energy policy . . . 3

1.3.2 Ship loading problem . . . 5

2 Mathematical properties **5**

2.1 Flow decomposition . . . 6

2.2 Removing nonzero lower bounds . . . 7

2.3 Removing infinite capacities . . . 7

2.4 Removing negative costs . . . 8

2.5 Making the network strongly connected . . . 9

2.6 Residual network . . . 9

2.7 Negative cycle optimality condition . . . 11

2.8 Reduced cost . . . 12

2.9 Reduced cost optimality condition . . . 13

3 Algorithms **13**

3.1 Finding a feasible flow . . . 14

3.2 Cycle-cancelling algorithm . . . 14

3.3 Successive shortest path algorithm . . . 15

4 Conclusion **18**

1 Introduction

Everywhere we look in our lives, networks are apparent. From electrical and power networks to manufacturing and distribution networks, there are infinite variations between the most physical and the most abstract networks. In all these problem domains, we wish to move some entity (electricity, a consumer product, a message) from one point to another in the underlying network, and to do so as efficiently as possible.

The minimum cost flow model is the most fundamental of all network flow problems. The problem is easy to state: we wish to determine a least cost shipment of a commodity through a network in order to satisfy demands at certain nodes from available supplies at other nodes. This model has a number of familiar and less familiar applications: the distribution of a product from plants to warehouses, or from warehouses to retailers; the routing of cars through a street network; or even the orthogonal drawing of a planar graph. In this section, we present a mathematical formulation of the minimum cost flow problem and some of its applications.

1.1 Mathematical formulation

Let $G = (N, A)$ be a directed graph, which is allowed to have parallel arcs. We denote by $N^+(v)$ the out-neighbourhood of v, and by $N^-(v)$ the in-neighbourhood of v (here, the neighbourhood is thought in terms of edges). We associated to each arc e three quantities:

- the cost $c(e)$ *per unit of flow* on that arc
- the capacity $u(e)$ that denotes the maximum amount that *can* flow on the arc
- the lower bound $l(e)$ that denotes the minimum amount that *must* flow on the arc

We also associate to each node v a number $b(v)$ representing its supply/demand. If $b(v) > 0$, node v is a *surplus node*; if $b(v) < 0$, node v is a *demand node* with a demand of $-b(i)$; and if $b(v) = 0$, node v is a *transshipment node*. The minimum cost flow problem can be formulated as an optimization problem as follows. The decision variables of the problem are the arc flows, which are denoted by $f(e)$ for each arc $e \in A$.

$$\text{Minimize} \sum_{e\in A} c(e)f(e)$$

subject to

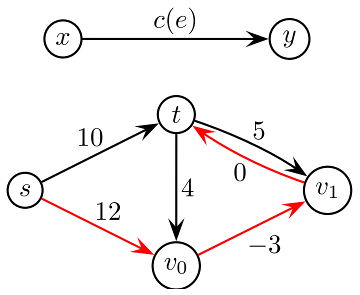
$$\sum_{e\in N^+(v)} f(e) - \sum_{e\in N^-(v)} f(e) = b(v) \quad \text{for all } v \in V$$

$$l(e) \leqslant f(e) \leqslant u(e) \quad \text{for all } e \in A$$

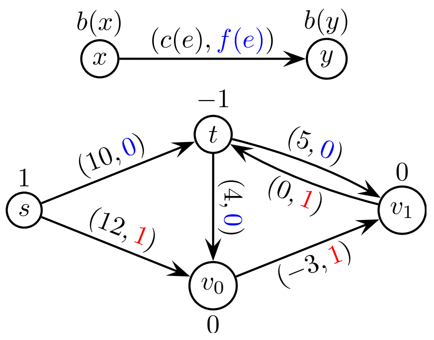
where $\sum_{v\in V} v(v) = 0$. Furthermore, we usually require that all quantity be *integers* because for most applications we can reach this assumption by multiplying all the quantities by a suitably large number. Moreover, irrational numbers generally need to be approximated by rational ones to represent them on a computer. We call the equality constraint on the flows the *mass balance constraint.*

1.2 Applications to graph theory

A number of special cases of the minimum cost flow happen to play a central role in graph theory and more generally in applications of network flows. We list two of them and explain why there are special cases of our general model. Furthermore, those two applications have a deep relation with the minimum cost flow problem since it shares common aspects with both of them. Finally, a number of minimum cost flow algorithms heavily rely on those problems in order to solve our general problem.



Shortest path graph (shortest path in red)



Network flow graph (minimum cost flow in red/blue)

Figure 1: Example of a shortest path problem and its mapping to the minimum cost flow model

1.2.1 Shortest path problem

The shortest path problem is one of the simplest of all network flow problems. For this problem, we wish to find a path of minimum cost (or length) from a specified *source node* s to another specified *sink node* t, assuming that each arc has an associated cost $c(e)$. In order to fit this model to our minimum cost flow problem, we set $b(s) = 1$, $b(t) = -1$ and $b(v) = 0$ for all other nodes. We further set $l(e) = 0$ and $u(e) = \infty$ for each arc in the graph. Intuitively, the try to send one unit of flow from s to t, and to do so at the minimum possible cost. One can then see and the unit of flow follows the shortest path. Figure 1 summarizes the transformation on an example.

In the case where we are interested in the shortest path from a source s to every other node in the graph, we set $b(s) = n - 1$ and $b(v) = -1$ for all other nodes, where $n = |V|$. We further set $l(e) = 0$ and $u(e) = \infty$. Intuitively, we send $n - 1$ units of flow from s and since all nodes demand exactly one unit (since $b(v) = -1$), we wish to send exactly one unit of flow to each node. One can then easily show that minimizing the total cost of all paths is equivalent to minimizing the cost of each path. Furthermore, the units of flow follow the shortest paths.

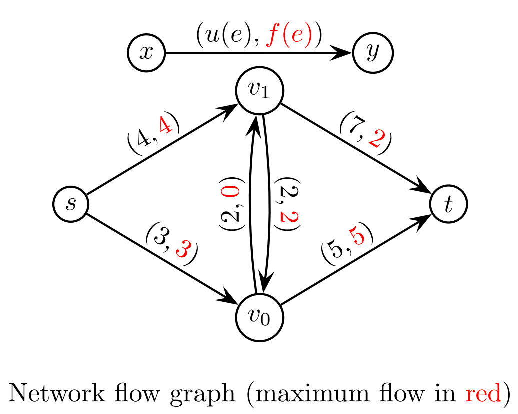
1.2.2 Maximum flow problem

The maximum flow problem is in a sense a complementary model to the shortest path problem. Instead of sending an unbounded amount of flow along a path of minimum cost, we wish to send a maximum amount of flow through the network regardless of the cost. Formally, the maximum flow problem seeks a feasible solution that sends the maximum amount of flow from a specified source s to a specified sink t, given a maximum bound on the amount we can send though an edge. We can formulate this problem as a minimum cost flow problem in the following manner. We set $b(v) = 0$ for all $v \in V$, $c(e) = 0$ for all $e \in A$ and keep the maximum bound $u(e)$. We further introduce an arc (t, s) with cost $c(t, s) = -1$ and a flow bound $u(t, s) = \infty$. Then the minimum cost flow maximizes the flow on the arc (t, s); but since any flow on arc (t, s) must travel from node s to t through the other arcs of network, we will maximize the flow from s to t in the original network. Figure 2 summarizes the transformation on an example.

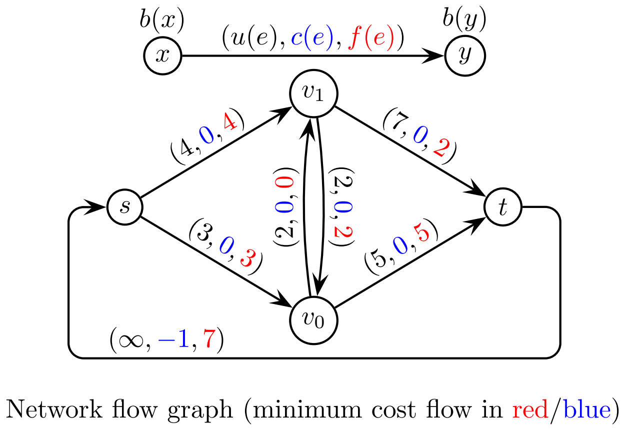
1.3 Applications to real world problems

1.3.1 Determining an optimal energy policy

Most countries or firms need to decide on an energy policy, *i.e.* how to utilize the available raw materials and transformations to satisfy their energy needs. Consider, for example, a country which has four raw materials: crude oil, coal, uranium and hydropower; and has four energy needs: electricity, domestic oil, petroleum and gas. The country has the necessary technology to convert crude oil into domestic oil or petroleum, coal into electricity and gas, and uranium and hydropower into electricity. Each available transformation comes at a specified cost per unit and a maximum amount of transformed material per year. Finally, the country can



Network flow graph (maximum flow in red)



Network flow graph (minimum cost flow in red/blue)

Figure 2: Example of a maximum flow problem and its mapping to the minimum cost flow model

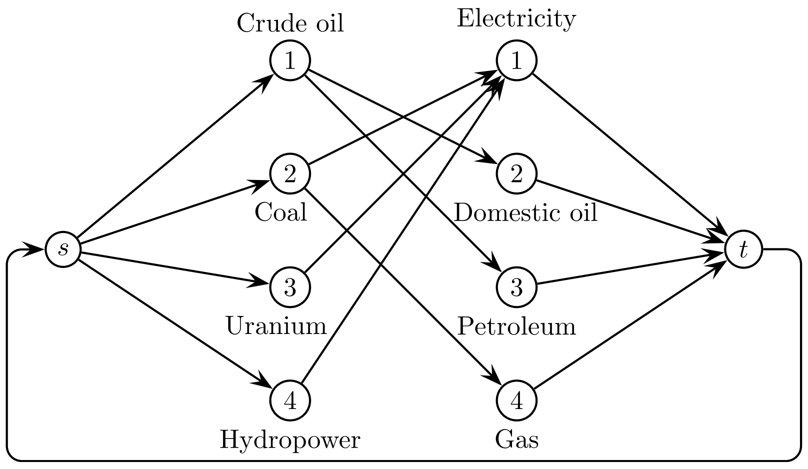


Figure 3: Energy policy problem formulated as a minimum cost flow problem

only produce a certain amount of each raw material at a specified cost per unit. The objective is to satisfy, at the least possible cost, a certain annual consumption level of various energy needs.

Figure 3 shows the formulation of the problem as a minimum cost flow problem. The network has three types of arcs:

- sources arcs $e = (s, m)$ from the source s to a raw material m with the cost $c(e)$ of producing the material m, an upper bound $u(e)$ on the production and a lower bound $l(e)$ on the production (some industry require a minimum production per year)
- conversion arcs $e = (m, n)$ from a raw material m to an energy need n with a the cost $c(e)$ of transformation per unit and upper bound $u(e)$ on the amount of transformed material per year
- sink arcs $e = (n, t)$ from a energy need n to the sink t with a cost $c(e) = 0$ and the lower bound $l(e)$ on the annual consumption of n

We further add an arc (t, s) with zero cost and unbounded capacity and set demand of nodes to zero: $b(v) = 0$. Then the minimum cost flow problem is exactly the energy policy problem as stated below. Notice that to formulate this problem in a more realistic way, we would need to introduce an *efficiency* value on the conversion edges since one unit of a raw material does not necessary produces one unit of something else; this is known as the *generalized flow problem*.

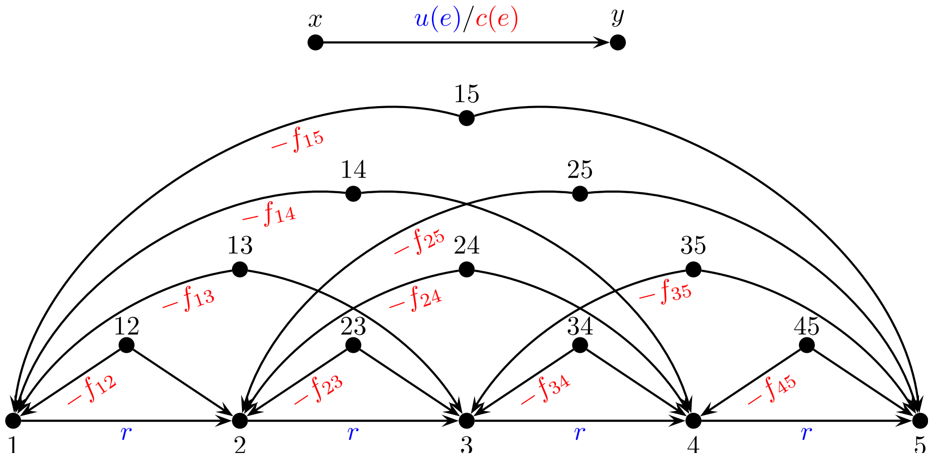


Figure 4: Ship loading problem formulated as a minimum cost flow problem

1.3.2 Ship loading problem

A small cargo company uses a container ship with a capacity to carry at most r freight containers. The ship sails on a route with several stops at ports in between. At these ports, containers may be unloaded and new containers may be loaded. At each port, there is a maximum amount b_{ij} of containers which is waiting to be shipped from port i to port $j > i$ (ports are numbered and visited in order from 1 to n). Let f_{ij} denote the income per container shipped from port i to port j. The goal of the cargo company is to plan how much to load at each port in order to maximize the income while never exceeding the total capacity of the ship.

Figure 4 shows the formulation of the problem as a minimum cost flow problem. The network has two types of nodes:

- *port nodes* v_i for $1 \leqslant i \leqslant n$ with balance $b(v_i) = -\sum_{j<i} b_{ji}$
- *shipment nodes* v_{ij} for $1 \leqslant i < j \leqslant n$ with balance $b(v_{ij}) = b_{ij}$; intuitively, v_{ij} represents the store of containers waiting to be shipped from i to j: there will either be shipped by the cargo company or by other means

The networks also has three types of arcs:

- *shipped by company arcs* $e = (v_{ij}, v_i)$ with capacity $u(e) = \infty$ and cost $c(e) = -f_{ij}$; intuitively, the flow on e represents the amount of containers shipped by the company from i to j: the income per container shipped can be seen as negative cost
- *shipped by other means arcs* $e = (v_{ij}, v_j)$ with capacity $u(e) = \infty$ and cost $c(e) = 0$; intuitively, the flow on e represents the amount of containers which are not shipped by the company from i to j, the income per container (and thus the cost) is 0
- *route arcs* $e = (vi, v_{i+1})$ with capacity $u(e) = r$ and cost $c(e) = 0$; intuitively, the flow on e represents the total amount of containers shipped from ports v_i to v_j by the company: it must not exceed the capacity of the ship

Then the minimum cost flow exactly describes a loading of the ship which maximizes the income of the company.

2 Mathematical properties

We study mathematical properties of network flows, in order to derive some theoretical as well as practical results. We are interested in two kinds of results: simplifications of the problem (zero lower bounds, positive costs), and necessary and sufficient conditions (feasible flow, minimum cost flow).

We will always assume from now on that the network contains no directed cycle of negative cost and infinite capacity; indeed if there is such a cycle, then the minimum cost flow is undefined since we can build flow of arbitrary small value.

We also define the following notations. If P is directed path or cycle, $\delta_e(P) = 1$ if $e \in P$ and $\delta_e(P) = 0$ otherwise. Similarly $\delta_v(P) = 1$ if v is on P. If P is a directed path, $\alpha(P)$ will refer to the starting node of P and $\beta(P)$ to the ending node. And if G is graph, $\mathcal{W}$ is the set of all directed cycles of G and $\mathcal{P}$ is the set of all directed paths of G. Furthermore, n will refer to the number of nodes and m to the number of arcs. We will denote by U the maximum finite capacity and by B the maximum demand/supply: $U = \max_e\{u(e) \mid u(e) < \infty\}$ and $B = \max_v |b(v)|$. We will also refer to C as the maximum cost: $C = \max_e |c(e)|$. Finally, if $e = (u, v)$ is an edge, we denote by $\widetilde{e}$ the reversed edge: $\widetilde{e} = (v, u)$.

2.1 Flow decomposition

When formulating network flow problems, we can adopt either one of two equivalent modelling approaches. We can define flows on arcs as usual or define flows on paths and cycles. Figure 5 shows those two models on a example.

Definition 2.1.1 (Cycle and path flow)**.** *A cycle and path flow f is a flow $\widehat{f}$ defined by:*

$$\widehat{f}(e) = \sum_{P\in\mathcal{P}} \delta_e(P)f(P) + \sum_{W\in\mathcal{W}} \delta_e(W)f(W)$$

We define the cost of a path or a cycle as $c(P) = \sum_{e\in P} c(e)$ and $c(W) = \sum_{e\in W} c(e)$. We define the cost of a cycle and path flow as

$$\sum_{P\in\mathcal{P}} f(P)c(P) + \sum_{W\in\mathcal{W}} f(W)c(W)$$

This definition shows that each cycle and path flow determines arcs flows uniquely. The following theorem shows that conversely, we can decompose any flow into a cycle and path flow.

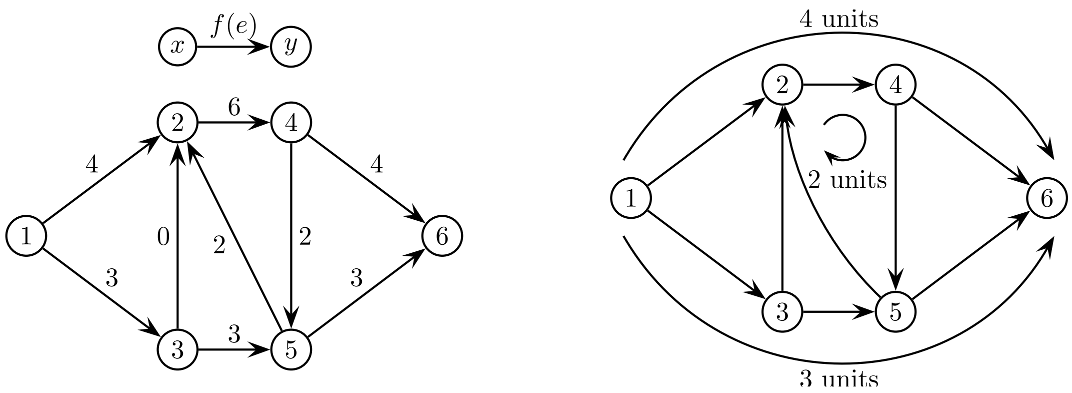


Figure 5: Two ways of representing flows in a network: using arc flows; or using path and cycle flows

Lemma 2.1.2. *The cost of a cycle and path flow f equals the cost of the flow $\widehat{f}$.*

Proof.

$$\begin{aligned}\sum_{e\in A} \widehat{f}(e)c(e) &= \sum_{e\in A}\left(\sum_{P\in\mathcal{P}} \delta_e(P)f(P) + \sum_{W\in\mathcal{W}} \delta_e(W)f(W)\right)c(e)\\ &= \sum_{P\in\mathcal{P}} f(P)\left(\sum_{e\in A}\delta_e(P)c(e)\right) + \sum_{W\in\mathcal{W}} f(W)\left(\sum_{e\in A}\delta_e(W)c(e)\right)\\ &= \sum_{P\in\mathcal{P}} f(P)c(P) + \sum_{W\in\mathcal{W}} f(W)c(W)\end{aligned}$$

□

Theorem 2.1.3 (Flow decomposition)**.** *Every flow f° can be represented as a cycle and path f such that $\widehat{f} = f^\circ$ with the following two properties:*

- *Every directed path with nonzero flow connects a supply node to a demand node*
- *At most $n+m$ paths and cycles have nonzero flow; out of these, at most m cycles have nonzero flow.*

Proof. We will give an algorithmic proof of the result. Suppose that v_0 is a supply node. Then there is some arc $e = (v_0, v_1)$ with nonzero flow. If v is a demand node, we stop with path $P = (v_0, v_1)$; otherwise the mass balance constraint at v_1 imposes that another arc (v_1, v_2) has nonzero flow. We repeat this argument until we either reach a demand node or encounter a previously visited node. One of these two cases will occur within n steps. In the former case we obtain a directed path P from a supply node to a demand node and in the latter case we obtain a directed cycle W. In either case, the path or the cycle only consists of nonzero flow arcs.

If we obtain a directed path P from v_0 to v_k, we set $f(P) = \min\{b(v_0), -b(v_k), \min_{e\in P} f(e)\}$ and redefine $b(v_0) = b(v_0) - f(P)$, $b(v_k) = b(v_k) + f(P)$ and $f(e) = f(e) - f(P)$ for every $e \in P$. If we obtain a directed cycle W, we set $f(W) = \min_{e\in W} f(e)$ and redefine $f(e) = f(e) - f(P)$ for every $e \in W$.

We repeat this process with the redefined problem until there are no more supply nodes, that is $b(v) = 0$ for all $v \in V$. Then we select any node with one outgoing edge with nonzero flow as a starting point and repeat the procedure. Since $b(v) = 0$ for all $v \in V$, we will go back to the starting point and find a directed cycle as before. We finish when $f = 0$ for the redefined problem.

Clearly, the original flow is the sum of flows on the paths and cycles we found. Now notice that each time we find a directed path, we redefine the demand or the supply to 0 or the flow on some arc to 0; similarly, each time we find a directed cycle, we redefine the flow on some arc to 0. As a consequence, we will terminate after identifying at most $n+m$ cycles and paths and identify at most m cycles. □

2.2 Removing nonzero lower bounds

In the most general statement of the problem, we allow arcs to have a nonzero lower bound $l(e)$ the arc flow $f(e)$. Although this is useful in some problems, it is usually simpler to assume that $l(e) = 0$ in the proofs or the algorithms.

To achieve this transformation, we consider another minimum cost flow problem with the same nodes and arcs. We replace the flow $f(e)$ in the original problem by $f'(e) + l(e)$ where $f'(e)$ is the flow in the new problem. The flow bound constraint then becomes $0 \leqslant f'(e) \leqslant u(e) - l(e)$. Making this substitution in the balance constraints for $e = (u, v)$ decreases $b(u)$ by $l(u)$ and increases $b(v)$ by $l(v)$. Finally, this transformation decreases the objective function by the constant $\sum_{e\in A} l(e)c(e)$.

Figure 6 illustrates this transformation on an arc. Then we can solve the original problem by solving the new problem and transforming the solution back to this original setting.

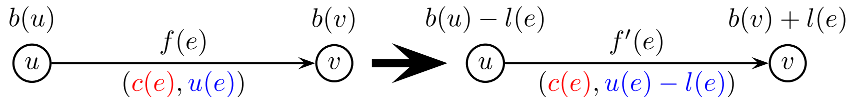


Figure 6: Removing nonzero lower bounds ($f(e) = f'(e) + l(e)$)

2.3 Removing infinite capacities

In many, if not most, minimum cost flow problem, there are some *uncapacitated arcs*, *i.e.* arcs with $u(e) = \infty$; such arcs are especially useful to simplify the problem formulation. However, in some transformations and in many algorithms, it is not desirable to have uncapacitated arcs since we do some arithmetic with capacities and flows. If an arc e has an infinite capacity, we wish to find a finite bound on the arc's flow. Hopeful, we usually know in advance an upper bound on the arc's flow; for example in the *ship loading problem*, the flow can't be greater than the capacity of the ship and the shipments waiting in some ports. In the general case, the following result provides an upper bound for such arcs.

Lemma 2.3.1. *There exists f°, a minimum cost flow such that* $f^\circ \leqslant \sum_{v\in V} |b(v)| + \sum_{\substack{e\in A\\ u(e)<\infty}} u(e)$

Proof. We already explained how we can transform the network in order to have zero lower bounds. We can thus assume that $l(e) = 0$, for all $e \in A$. Now apply theorem 2.1.3 to f°: we get a cycle and path flow f such that $\widehat{f} = f^\circ$. Thus if $e_0 \in A$,

$$f^\circ(e_0) = \sum_{P\in\mathcal{P}} \delta_{e_0}(P)f(P) + \sum_{W\in\mathcal{W}} \delta_{e_0}(W)f(W)$$

Now pick a cycle W such that $f(W) > 0$. Then we must have $c(W) \leqslant 0$ since f° is a minimum cost flow; otherwise we can set $f(W) = 0$ and because we have zero lower bounds, we obtain a feasible flow of strictly lesser cost than f°. Furthermore, if $c(W) = 0$ then we can set $f(W) = 0$ and obtain a feasible flow of the same cost. So we can assume that if $f(W) > 0$ then $c(W) < 0$.

But remember that f° is a minimum cost flow so if there is a cycle W such that $c(W) < 0$ then at least one edge e_W of W must be saturated, that is $f^\circ(e_W) = u(e_W) < \infty$ otherwise we could push some more flow on W and get a better flow. Let $\mathcal{W}^+ = \{W \mid f(W) > 0\}$ and $AW = \{e_W, W \in \mathcal{W}^+\}$. Then we have:

$$\sum_{W\in\mathcal{W}} \delta_{e_0}(W)f(W) \leqslant \sum_{W\in\mathcal{W}^+} f(W) \leqslant \sum_{e\in AW} f^\circ(e) \leqslant \sum_{e\in AW} u(e) \leqslant \sum_{\substack{e\in A\\ u(e)<\infty}} u(e)$$

Now rewrite the mass balance constraint with cycle and path costs for a node v:

$$b(v) = \sum_{e\in N^+(v)} f(e) - \sum_{e\in N^-(v)} f(e) = \sum_{\substack{P\in\mathcal{P}\\ \alpha(P)=v}} f(P) - \sum_{\substack{P\in\mathcal{P}\\ \beta(P)=v}} f(P)$$

But remember that from theorem 2.1.3, all directed path go from supply nodes to demand nodes, so for each v, either $\{P \mid \alpha(P) = v\} = \varnothing$ or $\{P \mid \beta(P) = v\} = \varnothing$. Thus we have:

$$|b(v)| = \sum_{\substack{P\in\mathcal{P}\\ \alpha(P)=v}} f(P) + \sum_{\substack{P\in\mathcal{P}\\ \beta(P)=v}} f(P)$$

So finally:

$$\sum_{P\in\mathcal{P}} \delta_{e_0}(P)f(P) \leqslant \sum_{P\in\mathcal{P}} f(P) \leqslant \sum_{v\in V} \sum_{\substack{P\in\mathcal{P}\\ \alpha(P)=v}} f(P) \leqslant \sum_{v\in V} |b(v)|$$

Putting things together proves the result. □

2.4 Removing negative costs

In many problem formulations it is convenient to work with negative costs; indeed minimizing a negative cost quantity is equivalent to maximizing a positive profit. However, it is much more convenient to only allow nonnegative costs in the proofs and algorithms. In this transformation, we need to handle *uncapacitated* arcs carefully. Thus, we rely on lemma 2.3.1 to ensure that all capacities are finite.

In this transformation we consider another minimum cost flow problem with the same nodes. All the arcs with nonnegative cost are kept in the new problem and for these $f(e) = f'(e)$ where f' is the flow in the new network. On the contrary, for each arc $e = (u, v)$ with negative cost, we add an edge $e' = \widetilde{e}$ in the new problem, with cost $c'(e') = -c(e)$ and we replace the flow $f(e)$ in the original network by $u(e) - f'(e)$. To account for the change in balance, we decrease $b'(u)$ by $u(e)$ and increase $b'(v)$ by $u(e)$.

Figure 7 illustrates this transformation on an arc. Then we can solve the original problem by solving the new problem and transforming the solution back to this original setting. The intuition of this transformation is that instead of sending $f(e)$ units of flows and paying $c(e)$ each, we beginning by sending $u(e)$ and then $f'(e)$ account for the diminution of the flow. Then decreasing the flow costs $-c(e)$. And we change the demand/supply of the nodes to make everything consistent.

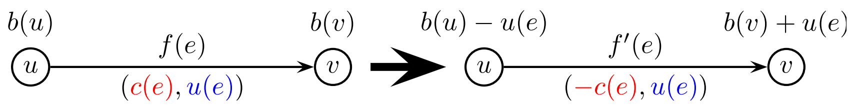


Figure 7: Working with positive costs ($f(e) = u(e) - f'(e)$)

2.5 Making the network strongly connected

When working with reduced costs, it will be easier to assume that there is path between each pair of nodes; that is, that the network is strongly connected. And even stronger, we will want that there is an uncapacitated path between each pair of nodes. Since this hypothesis is hardly verified in general, we need to explain how to transform the network to make it strongly connected. To do so, we will assume that the network has finite capacities, which is always possible thanks to transformation 2.3. Notice that we will add arcs with infinite capacities in this transformation, so one will eventually have to make these finite but these transformations will not “loop”.

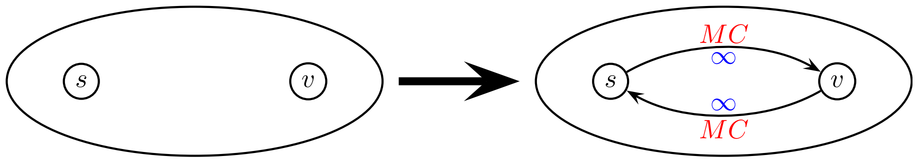


Figure 8: Making the network strongly connected ($c(e) = MC, u(e) = \infty$)

Now define $M = \sum\{|b(v)|, v \in V\} + \sum\{u(e), e \in A \,|\, u(e) < \infty\}$. From 2.3.1, we know that there is a minimum cost flow f^* which satisfies $\forall e \in A, f^*(e) \leqslant M$. Now, fix a node $s \in V$ and for each $v \in V$, add an arc $e = (s, v)$ and an arc $e' = (v, s)$ with cost $c(e) = c(e') = 1 + MC$ and capacity $u(e) = u(e') = \infty$. After this transformation, the network clearly satisfies the wanted property: there is an uncapacitated path between each pair of nodes. Figure 8 illustrates the transformation on an example. But what about the minimum cost flows ? The following lemma shows that they remain the same.

Lemma 2.5.1. *The minimum cost flows remain the same after the transformation.*

Proof. Let A' be the set of arcs after the transformation. Let f° be a minimum cost flow. Then by definition of f° and f^* we have:

$$\sum_{e \in A'} f^\circ(e)c(e) = \sum_{e \in A} f^*(e)c(e)$$

Now suppose that there is an arc $e_0 \in A' \setminus A$ such that $f^\circ(e_0) > 0$, *i.e.* there is a minimum cost flow which use one of the new arcs. Then we must have (because all the data are integral):

$$\sum_{e \in A'} f^\circ(e)c(e) \geqslant f^\circ(e_0)c(e_0) \geqslant c(e_0) = 1 + MC > MC \geqslant \sum_{e \in A} f(e)c(e)$$

Which contradicts the optimality of f°. Thus f° does not use any new arc. □

2.6 Residual network

When implementing network flow algorithms, we are not thinking in terms of absolute flow but rather in terms of *incremental* flow. To do so, we define the *residual network*, that gives information about how we can push or extract some flow into or from the network.

The idea behind the residual network is the following. Suppose that an edge e carries $f^\circ(e)$ units of flow, then we can still push up to $u(e) - f^\circ(e)$ units of flow on this edge. But we can also extract up to $f^\circ(e)$ units of flow from it; to do so we can push up to $f^\circ(e)$ units of flow on $\widetilde{e}$, which amounts to cancelling the existing flow on e. Finally notice that pushing a unit of flow on e costs $c(e)$ and that pushing one unit of flow on $\widetilde{e}$ costs $-c(e)$; since we are removing some flow.

Formally, the residual network with respect to a given flow f° is a network with the same nodes defined as follows. For each arc e in the original network, we add an arc e with cost $c(e)$ and *residual capacity* $r(e) = u(e) - f^\circ(e)$; and the arc $\widetilde{e}$ with cost $-c(e)$ and residual capacity $f^\circ(e)$. The residual network consists **only of arcs with a positive residual capacity**. Thus we delete arcs with a zero capacity. We denote by $G(f^\circ)$ this network.

Figure 9 illustrates the construction for an arc. Notice that the residual graph may have parallel arcs even though the original network doesn't have parallel ones.

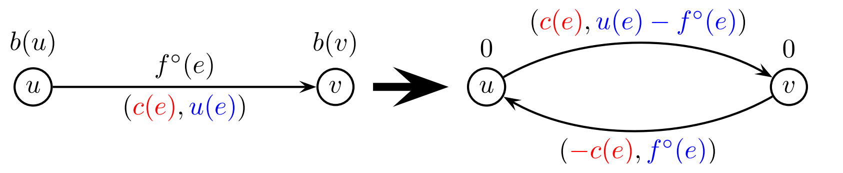


Figure 9: Construction of the residual network

The main interest of the residual network is that it makes it easier to state properties while preserving a one-to-one correspondence between feasible flows in the two networks that preserves the cost of the solutions. We state a more precise version of this result in the following theorem.

Theorem 2.6.1. *Let f° be a feasible flow of a network N and $G = G(f^\circ)$. Then a flow f is feasible in N if and only if the corresponding flow f' defined by $f'(e) - f'(\widetilde{e}) = f(e) - f^\circ(e)$ and $f'(e)f'(\widetilde{e}) = 0$ is a feasible flow in G. Furthermore,*

$$\sum_{e\in A} c(e)f(e) = \sum_{e\in A_G} c'(e)f'(e) + \sum_{e\in A} c(e)f^\circ(e)$$

where c is the cost in N and c' the cost in G.

Proof. First notice that the flow f' is well-defined. Indeed, the condition $f'(e)f'(\widetilde{e}) = 0$ implies that either $f'(e) = 0$ or $f'(\widetilde{e}) = 0$. But if $f'(e) = 0$ then $f'(\widetilde{e}) = f^\circ(e) - f(e)$; and if $f'(\widetilde{e}) = 0$ then $f'(e) = f(e) - f^\circ(e)$. Thus, it only depends on the sign of $f(e) - f^\circ(e)$.

$\boxed{\Rightarrow}$ First assume that f is a feasible flow and define f' as above. Then f' must satisfy the bound constraints, there are two cases to check:

- if $f(e) \geqslant f^\circ(e)$ then $0 \leqslant f'(e) = f(e) - f^\circ(e) \leqslant u(e) - f^\circ(e) = r(e)$ because $f(e) \leqslant u(e)$. Furthermore $f'(\widetilde{e}) = 0$ so it satisfies the bound constraint
- if $f(e) \leqslant f^\circ(e)$ then $0 \leqslant f'(\widetilde{e}) = f^\circ(e) - f(e) \leqslant f^\circ(e) = r(\widetilde{e})$. Furthermore, $f'(e) = 0$ so it satisfies the bound constraint

Then, we can check that the mass balance constraint is satisfied with a simple calculus:

$$\begin{aligned}\sum_{e\in N_G^+(v)} f'(e) - \sum_{e\in N_G^-(v)} f'(e) &= \sum_{e\in N^+(v)} (f'(e) - f'(\widetilde{e})) - \sum_{e\in N^-(v)} (f'(e) - f'(\widetilde{e})) \\ &= \sum_{e\in N^+(v)} (f(e) - f^\circ(e)) - \sum_{e\in N^-(v)} (f(e) - f^\circ(e)) = b(v) - b(v) = 0\end{aligned}$$

Finally, we can link the relationship on the cost as follows (and then summing over the arcs).

$$c'(e)f'(e) + c'(\widetilde{e})f'(\widetilde{e}) = c(e)(f'(e) - f'(\widetilde{e})) = c(e)(f(e) - f^\circ(e)) = c(e)f(e) - c(e)f^\circ(e)$$

$\boxed{\Leftarrow}$ Now assume that f' is a feasible flow. Then $f(e) = (f'(e) - f'(\widetilde{e})) + f^\circ(e)$ and f must satisfy the bound constraints, there are two cases to check:

- if $f'(e) = 0$ since $0 \leqslant f'(\widetilde{e}) \leqslant r(\widetilde{e}) = f^\circ(e)$, we have $0 \leqslant f(e) \leqslant f^\circ(e) \leqslant u(e)$

- if $f'(\widetilde{e}) = 0$ since $0 \leqslant f'(e) \leqslant r(e) = u(e) - f^\circ(e)$, we have $0 \leqslant f^\circ(e) \leqslant f(e) \leqslant u(e)$

Then we can easily check the mass balance constraint is satisfied:

$$\begin{aligned}\sum_{e\in N^+(v)} f(e) - \sum_{e\in N^-(v)} f(e) &= \sum_{e\in N^+(v)} (f'(e) - f'(\widetilde{e}) + f^\circ(e)) - \sum_{e\in N^-(v)} (f'(e) - f'(\widetilde{e}) + f^\circ(e)) \\ &= \sum_{e\in N^+(v)} f^\circ(e) - \sum_{e\in N^-(v)} f^\circ(e) + \sum_{e\in N^+_G(v)} f'(e) - \sum_{e\in N^-_G(v)} f'(e) \\ &= b(v) + 0 = b(v)\end{aligned}$$

Finally, the calculus about cost holds in both cases, so there is no need to redo it.

□

2.7 Negative cycle optimality condition

Thanks to the properties of the residual network, we are now able to state a first optimality condition for the minimum flow problem. As we will see, this condition give rise to very simple (but inefficient algorithm) for solving the problem.

Theorem 2.7.1. *A feasible flow f^* is an optimal solution of the minimum cost flow problem if and only if $G(f^*)$ contains no negative cost directed cycle.*

Proof. $\boxed{\neg \Leftarrow \neg}$ Assume that f^* is an feasible flow and that $G(f^*)$ contains a negative cost directed cycle W of cost $c'(W) < 0$ and maximum flow $f'(W) > 0$ on it. Then f^* cannot be optimal because we can augment it along W and decrease its value by $-c'(W)f'(W)$. Formally, consider the residual flow $f'(e) = \delta_e(W)f(W)$. Then f' is a feasible flow in $G(f^*)$ by hypothesis and by applying theorem 2.6.1 we get a flow feasible f which is strictly better than f^* and thus yields a contradiction:

$$\sum_{e\in A} c(e)f(e) - \sum_{e\in A} c(e)f^*(e) = \sum_{e\in A_G} c'(e)f'(e) = c'(W)f'(W) < 0$$

$\boxed{\neg \Rightarrow \neg}$ Assume that f^* is a feasible flow and that $G(f^*)$ contains no negative cost directed cycle. Let f° be any feasible flow. We will show that f^* has a lesser cost than f°. Apply theorem 2.6.1 to f° in $G(f^*)$. Then we get a residual flow f'. Now apply the flow decomposition theorem (2.1.3) to f' in $G(f^*)$. We get a cycle and path flow x such that $\widehat{x} = f'$. But recall that the mass balance constraint in $G(f^*)$ implies that all nodes are transshipment nodes. Thus x is only a cycle flow. But since $G(f^*)$ contains no negative cost directed cycle, the cost of any cycle must be nonnegative and thus x has a nonnegative cost:

$$\sum_{W\in\mathcal{W}(G(f^*))} c'(W)x(W) \geqslant 0$$

Now, using the cost relationship of a cycle and path flow (2.1.2) we get that:

$$\sum_{e\in A_G} c'(e)f'(e) = \sum_{e\in A_G} c'(e)\widehat{x}(e) = \sum_{W\in\mathcal{W}(G(f^*))} c'(W)x(W) \geqslant 0$$

And finally, using the cost relationship of the residual network network, we get that:

$$\sum_{e\in A} c(e)f^\circ(e) - \sum_{e\in A} c(e)f^*(e) = \sum_{e\in A_G} c'(e)f'(e) \geqslant 0$$

Thus, f^* is an optimal flow. □

2.8 Reduced cost

In many minimum cost flow algorithms, we measure the cost of an arc related to some cost associated with the nodes. These costs are typical intermediate data of the algorithm. Assume that we associate to each node $v \in V$ a *potential* $\pi(v)$. We then define the *reduced cost* with respect to π as c^π defined by:

$$c^\pi(e) = c(e) + \pi(v) - \pi(u) \qquad \text{if } e = (u, v)$$

In those algorithms, the reduced cost replaces the cost, especially in the residual network. Thus, it is crucial to understand the relationship between c and c^π, with respect to the minimum flow problem. We first relate the objective function for c with the objective function for c^π:

Lemma 2.8.1. *A minimum cost flow problem has the same optimal solutions if we replace the cost c by c^π. Furthermore, for any feasible flow f we have:*

$$\sum_{e\in A} f(e)c(e) - \sum_{e\in A} f(e)c^\pi(e) = \sum_{v\in V} \pi(v)b(v)$$

Proof. First, we establish the relationship on the cost:

$$\begin{aligned}
\sum_{e\in A} f(e)c^\pi(e) &= \sum_{e\in A} f(e)c(e) + \sum_{e=(u,v)\in A} f(e)\pi(v) - \sum_{e=(u,v)\in A} f(e)\pi(u) \\
&= \sum_{e\in A} f(e)c(e) + \sum_{v\in V} \pi(v) \sum_{e\in N^-(v)} f(e) - \sum_{u\in V} \pi(u) \sum_{e\in N^+(v)} f(e) \\
&= \sum_{e\in A} f(e)c(e) + \sum_{v\in V} \pi(v) \left(\sum_{e\in N^-(v)} f(e) - \sum_{e\in N^+(v)} f(e) \right) \\
&= \sum_{e\in A} f(e)c(e) - \sum_{v\in V} \pi(v)b(v)
\end{aligned}$$

Thus the total decrease in the objective function does not depend on the flow. Therefore if f minimizes the objective function for c, it minimizes the objective function for c^π. □

We can also study the effect of the reduced cost on the cycle and paths flows that we introduced for the flow decomposition:

Lemma 2.8.2. *For any network flow with cost function c, for any potential function π we have:*

1. *For any directed cycle W, $\sum_{e\in W} c(e) = \sum_{e\in W} c^\pi(e)$*
2. *For any directed path from u to v, $\sum_{e\in P} c^\pi(e) = \sum_{e\in P} c(e) + \pi(v) - \pi(u)$*

Proof. trivial □

The introduction of reduced costs is related to another network flow problem, namely the shortest path problem. In this context, the potential can seen as the *potential shortest path.* We now present a result to link the shortest path problem to the use of potentials and give an intuition about minimum cost flow:

Lemma 2.8.3. *Assume there is a source node $s \in V$ such that every node $v \in V$ is accessible from s. Let d be a potential function and $\pi = -d$. Then if there is no negative cost cycle in the network, the following two conditions are equivalent:*

(i) $d(v)$ is the shortest path distance from s to v with respect to c

(ii) $d(s) = 0$ and $c^\pi(e) \geqslant 0 \qquad \forall e \in A$

Proof. We will implicitly use the fact that d is well-defined, that is, there is no negative cost cycle.
$\boxed{(i) \Rightarrow (ii)}$ Let $e = (x, v) \in A$. Now write the fact that $d(v)$ is the shortest path distance from s to v with respect to c:

$$d(v) \leqslant d(x) + c(e) \quad \Rightarrow \quad c(e) + d(x) - d(v) \geqslant 0 \quad \Rightarrow \quad c^{\pi}(e) = c^{-d}(e) \geqslant 0$$

$\boxed{\neg(i) \Rightarrow \neg(ii)}$ There are two cases: either $d(s) \neq 0$ and in this case d trivially cannot be the shortest path distance (since the shortest path distance from s to s must be 0). Either $\exists e_0, c^{\pi}(e_0) < 0$. By writing $e_0 = (x_0, v_0)$ we get that:

$$c^{\pi}(e_0) < 0 \quad \Rightarrow \quad c(e_0) + d(x_0) - d(v_0) < 0 \quad \Rightarrow \quad d(x_0) + c(e_0) < d(v_0)$$

Which contradicts the fact that d is the shortest path distance function. □

2.9 Reduced cost optimality condition

We will now try to merge the result about shortest path reduced cost (2.8.3) and negative cycle optimality condition (2.7.1). To do so, we will have to assume that the network is strongly connected and there is an "uncapacitated" path between each pair of vertices, a property which does not incur any loss of generality, thanks to section 2.5. By "uncapacitated" we mean high enough, because we actually want all capacities to be finite but want to be able to push any reasonable amount of flow.

The intuition of the reduced cost optimality condition is the following. We know that a flow is optimal if there is no negative cost cycle in the residual network. We will try to reformulate the last part. If there is no negative cost cycle in the residual network, it means that the shortest path distance is well defined with respect to the cost function. On the contrary, if there is a negative cost cycle, the shortest path distance is not well-defined. We will express the "well-definedness" of the shortest path in terms of reduced costs, thanks to the property 2.8.3. This give rise to the following theorem.

Theorem 2.9.1 (Reduced cost optimality condition). *A feasible flow f^* is an optimal solution of the minimum cost flow problem if and only if $\exists \pi$ a potential function satisfying the* reduced cost optimality condition *in the residual network $G(f^*)$:*

$$c^{\pi}(e) \geqslant 0 \quad \forall e \in A(G(f^*))$$

Proof. $\boxed{\Rightarrow}$Assume that f^* is an optimal solution. By the negative cycle optimality condition theorem (2.7.1), there is no negative cost cycle in $G(f^*)$. Thus the shortest path distance $d(\cdot)$ in $G(f^*)$ is well-defined. Now apply lemma 2.8.3 to $G(f^*)$ and we get that the potential $\pi = -d$ satisfy the reduced cost optimality condition.

$\boxed{\Leftarrow}$ Assume that there exists a potential function π such that the reduced cost optimality condition is satisfied. We will show that any cycle in $G(f^*)$ has nonnegative cost. Then by the negative cycle optimality condition theorem (2.7.1), we will conclude that f^* is optimal. Let W be a directed cycle in $G(f^*)$. By lemma 2.8.2, we can link the cost of W in $G(f^*)$ to its reduced cost and because all the reduced costs are nonnegative, the cycle must have a nonnegative cost:

$$\sum_{e \in W} c(e) = \sum_{e \in W} c^{\pi}(e) \geqslant 0 \qquad \text{because } c^{\pi}(e) \geqslant 0,\ \forall e \in A(G(f^*))$$

□

3 Algorithms

In this section, we describe a simple algorithm to solve the minimum cost flow problem in a very inefficient way. There are many more algorithms to solve this problem, some weakly polynomial time ones and even some strongly polynomial time ones but we refer to [1] for more details.

In this section, we assume that all networks satisfy the following properties:

- The network has no directed cycle of negative cost and infinite capacity, as before
- All capacities, costs and demands/supplies are integral, as before
- All lower bounds are zero; we refer to 2.2 for more details
- All capacities are finite; we refer to 2.3 for more details
- All arcs costs are nonnegative; we refer to 2.4 for more details
- There is an "uncapacitated" path between each pair of vertices; we refer to 2.5 for more details

One should notice that these assumptions imply no loss of generality and do not change the complexity of the algorithm.

3.1 Finding a feasible flow

A large class of minimum flow algorithms start by computing a feasible flow and then improve it until it reaches optimality. Finding a feasible flow is not a trivial problem because of the mass balance constraint. To determine a feasible flow, we will reduce the problem to a maximum flow problem.

Consider the network G with the same nodes and arcs as the original network, but without any costs. Now add two special nodes s^* and t^* and add arcs as follows. For each $v \in V$, if $b(v) > 0$ then add an arc (s^*, v) with capacity $b(v)$, if $b(v) < 0$ then add an arc (v, t^*) with capacity $-b(v)$. Now solve a maximum flow problem from s^* to t^* in the new network. If the maximum flow saturates all the source arcs, then the original network has a feasible flow; otherwise it is infeasible. Furthermore, to obtain a feasible flow from the maximum flow, we just need to forget about the arcs we added.

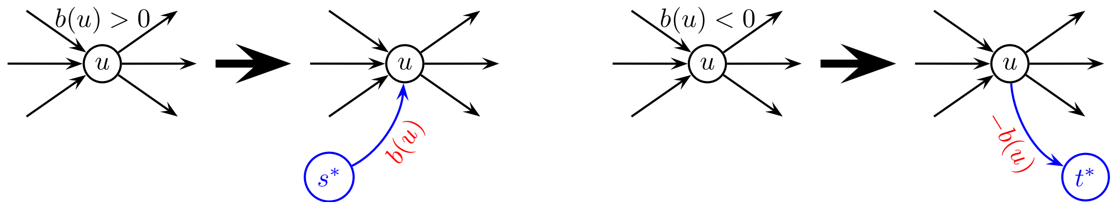


Figure 10: Transformation from the feasible flow to the maximum flow problem

Figure 10 illustrates the transformation on a node. The following lemma gives a mathematical basis to this transformation. Algorithm 1 describes the pseudo-code of such an algorithm.

Lemma 3.1.1. *The original network has a feasible flow if and only if every maximum flow of G saturates all the source arcs. Furthermore, if f^* is a maximum flow of G. Then f defined by $f = f_{|V}$ is a feasible flow of the original network.*

Proof. trivial. □

3.2 Cycle-cancelling algorithm

The negative cycle optimality condition stated in theorem 2.7.1 suggests a natural algorithm to find minimum cost flow, which is commonly called the *cycle-cancelling algorithm*. This algorithm starts with a feasible flow and tries to find a negative cost cycle in the residual graph. When no such cycle exists, we know we have an optimal flow; otherwise, we augment the flow along the cycle, decrease the cost of the flow and repeat the procedure. Algorithm 2 describes the pseudo-code of this an algorithm. Theorem 3.2.1 gives an upper bound on complexity of this algorithm.

Theorem 3.2.1. *Algorithm 2 has running time $O(nm^2CU)$.*

Proof. Consider the following points:

Function 1: FeasibleFlow(N, b, u)

```
Data: A network $N = (V, A)$; a demand/supply function $b$; a capacity function $u$
Result: A feasible flow or $\varnothing$
begin
    $V \leftarrow V \cup \{s^*, t^*\}$
    foreach $v \in V$ do
        if $b(v) > 0$ then
            $A \leftarrow A \cup \{(s^*, v)\}$
            $u(s^*, v) \leftarrow b(v)$
        else if $b(v) < 0$ then
            $A \leftarrow A \cup \{(v, t^*)\}$
            $u(v, t^*) \leftarrow -b(v)$
    $f \leftarrow$MaximumFlow($N, u$)
    if $f$ saturates all source arcs then return $f$
    else return $\varnothing$
```

Function 2: CycleCanceling(N, b, u)

```
Data: A network $N = (V, A)$; a demand/supply function $b$; a capacity function $u$; a cost function $c$
Result: A minimum cost flow or $\varnothing$
begin
    $f \leftarrow$FeasibleFlow($N, b, u$)
    if $f = \varnothing$ then return $\varnothing$
    $(G, c', r) \leftarrow$ResidualGraph($N, c, u$)
    $W \leftarrow$NegativeCostCycle($G, c'$)
    while $W \neq \varnothing$ do
        $\delta \leftarrow \min\{r(e), e \in W\}$
        $f \leftarrow$AugmentFlow($f, \delta, W, c'$)
        $(G, c', r) \leftarrow$ResidualGraph($N, c, u$)
        $W \leftarrow$NegativeCostCycle($G, c'$)
    return $f$
```

- Finding a feasible flow reduces to finding a maximum flow which can be done in $O(n^2m)$ time (using a variant of Ford and Fulkerson algorithm which augments along shortest paths, see [2] or [1])
- The initial flow has cost at most mCU: since $f(e) \leqslant U$ and $c(e) \leqslant C$, $c(e)f(e) \leqslant CU$ and thus $\sum_{e \in A} c(e)f(e) \leqslant mCU$
- The minimum cost flow has cost at least 0 since all costs are nonnegative
- Each iteration decreases the flow cost by at least 1 since all the data are integral
- Finding a negative cost directed cycle in a graph can be done in $O(nm)$ time (using the Bellman and Ford algorithm, see [2] or [1])

Thus, the algorithm executes at most $O(mCU)$ times a $O(nm)$ algorithm, plus a $O(n^2m)$ algorithm. Which gives a total $O(nm^2CU)$ running time. $\square$

Notice that this algorithm only has a pseudo-polynomial running time since it depends on C and U instead of $\log C$ and $\log U$.

3.3 Successive shortest path algorithm

The cycle-cancelling algorithm works by finding a feasible flow and decreasing the total cost until a certain condition is satisfied and implies optimality (namely the negative cycle condition). On the contrary, we will now present an algorithm based on the reduced cost optimality condition which works the other way around. That is, it will maintain a *pseudoflow* which always satisfy the optimality condition and will push flow on it until it becomes a real flow. This pseudoflow will satisfy the capacity constraints but will violate the mass-balance constraint.

Definition 3.3.1 (Pseudoflow)**.** *A* pseudoflow *is a function $f : A \to \mathbb{R}^+$ satisfying only the capacity constraints. If f is a pseudoflow, we define the* imbalance *at a node v as:*

$$e(v) = b(v) + \sum_{e \in N^-(v)} f(e) - \sum_{e \in N^+(v)} f(e) \qquad \forall v \in V$$

If $e(v) > 0$, we say that $e(v)$ is the excess *of v; and if $e(v) < 0$, we say that $-e(v)$ is the* deficit *of v; otherwise v is said to be* balanced*.*

Notice that since the total balance of the network is 0, if the network has an excess node, it must have a deficit node. We will now apply the reduced cost optimality condition to the concept of pseudoflow to see how we can derive an algorithm. The idea is that given a pseudoflow and potential function satisfying the reduced cost optimality condition, we will push some flow into the network and build a new potential function satisfying the reduced cost optimality condition.

Lemma 3.3.2. *Assume that a pseudoflow f satisfies the reduced cost optimality condition for some potential π. Let $d(\cdot)$ be the shortest path distance function from a node $s \in V$ to all other nodes in the residual graph $G(f)$ with respect to the reduced cost c^π. Then, the following two properties hold:*

(i) f satisfies the reduced cost optimality condition with respect to the potential $\pi' = \pi - d$

(ii) for each arc e in a shortest path from s to any other node, $c^{\pi'}(e) = 0$

Proof. Since $d(\cdot)$ is the shortest path distance function in $G(f)$ with respect to cost c^π, we can apply lemma 2.8.3 which yields that for each arc $e = (u, v) \in A(G(f))$:

$$\begin{aligned} (c^\pi)^{-d}(e) \geqslant 0 \quad &\Rightarrow \quad c^\pi(e) + (-d)(v) - (-d)(u) \geqslant 0 \\ &\Rightarrow \quad c(e) + (\pi(v) - d(v)) - (\pi(u) - d(u)) \geqslant 0 \\ &\Rightarrow \quad c^{\pi'}(e) \geqslant 0 \end{aligned}$$

Now if $e = (u, v)$ is in a shortest path from s to any node with respect to cost c^π, we have by definition:

$$d(v) = d(u) + c^\pi(e) \quad \Rightarrow \quad c^{\pi'}(e) = 0$$

□

As explained before, this lemma implies that we can augment the pseudoflow along a shortest path and still get a pseudoflow satisfying the reduced cost optimality condition. The next lemma formalizes this intuition. Recall that $r(e)$ is the residual capacity of e in the residual graph $G(f)$.

Lemma 3.3.3. *Assume that a pseudoflow f satisfies the reduced cost optimality condition for some potential π. Let $d(\cdot)$ be the shortest path distance function from a node $s \in V$ to all other nodes in the residual graph $G(f)$ with respect to the reduced cost c^π. Let P be a shortest path from s to any node v_P.*
Define $\mu = \min(e(s), -e(v_P), \min_{e \in P} r(e))$. Define f' by $f'(e) = f(e) + \delta_e(P)\mu$. Then f' is a pseudoflow that satisfies the reduced cost optimality condition for $\pi' = \pi - d$.

Proof. The fact that f' is a pseudoflow is a direct consequence of the choice of μ and the fact that P is a path in $G(f)$. Now, from lemma 3.3.2 we know that for each arc $e \in P$, since P is a shortest path, $c^{\pi'}(e) = 0$. Thus e satisfies the reduced cost optimality condition with respect to π'. But recall that augmenting the flow along e might add the arc $\widetilde{e}$ to the residual graph. Hopefully, $c^{\pi'}(\widetilde{e}) = -c^{\pi'}(e) = 0$. Thus, if it exists, $\widetilde{e}$ also satisfies the reduced cost optimality condition with respect to π'. □

We now have all the ingredients to describe the successive shortest path algorithm. Lemma 3.3.3 gives the key idea of the algorithm. The potential will be at the centre of the algorithm; we will maintain at the same time a pseudoflow f and a potential π. At each step of the algorithm, we will augment f along a shortest path with respect to the reduced cost c^π. By doing so, we will reduce the excesses and deficits in the network until all nodes are balanced. At this point, the pseudoflow will become a flow which satisfies the reduced cost optimality condition, and from theorem 2.9.1 we will know that this flow is optimal. Algorithm 3 describes the pseudo-code of this algorithm. Theorem 3.3.4 gives an upper bound on complexity of this algorithm.

Function 3: SuccessiveShortestPath(N, b, u)

```
Data: A network $N = (V, A)$; a demand/supply function $b$; a capacity function $u$; a cost function $c$
Result: A minimum cost flow or $\varnothing$
begin
    $f(e) \leftarrow 0$ for all $e \in A$
    $\pi(e) \leftarrow 0$ for all $e \in A$
    $E \leftarrow \{v \in V \mid e(v) > 0\}$
    $D \leftarrow \{v \in V \mid e(v) < 0\}$
    while $E \neq \varnothing$ do
        pick $s \in E$ and $v \in D$
        $(G, c', r) \leftarrow$ResidualGraph($N, c^\pi, u$)
        $d \leftarrow$ShortestPathDistance($G, c^\pi, s$)
        $P \leftarrow$ShortestPath($G, c^\pi, d, s, v$)
        $\mu \leftarrow \min(e(s), -e(v), \min_{e \in P} r(e))$
        $f \leftarrow$AugmentFlow($f, \mu, P, c'$)
        $\pi \leftarrow \pi - d$
        $E \leftarrow \{v \in V \mid e(v) > 0\}$
        $D \leftarrow \{v \in V \mid e(v) < 0\}$
    return $f$
```

Theorem 3.3.4. *Algorithm 3 has running time $O(nm\log(n)B)$.*

Proof. Consider the following points:

- Initialization steps are negligible compared to the cost of the loop body
- The initial pseudoflow has a total excess ($i.e$ $\sum_{v \in E} e(v)$) of at most nB.
- The minimum cost flow has a total excess of 0
- Each iteration decreases the excess of some node (and thus the total excess) by at least 1 since all the data are integral
- Finding the shortest path distance in a directed graph with nonnegative costs (since c^π satisfies the reduced cost optimality condition) can be done in $O(m\log(n))$ time (using the Dijkstra's algorithm with a binary heap, see [2] or [1])

Thus, the algorithm executes at most $O(nB)$ times a $O(m\log(n))$ algorithm. Which gives a total $O(nm\log(n)B)$ running time. □

Notice that this algorithm only has a pseudo-polynomial running time since it depends on B instead of $\log B$. However, this is already much better that the cycle cancelling algorithm in general. Furthermore, using some capacity scaling and doing some very small modifications to the algorithm, it is possible to obtain a weakly-polynomial algorithm. The idea is that instead of augmenting along a shortest path between any pair of vertices, we first augment the flow between vertices with high excess and deficit.

4 Conclusion

We saw a mathematical formulation of the minimum cost flow problem and we saw that it is a natural generalisation of the maximum flow problem and the shortest path problem. We also gave the intuition that a number of real-life problems can easily be stated in terms of minimum cost flow. We then studied some properties of the minimum cost flows in a network. In particular, we gave two different optimality conditions which naturally require the introduction of the residual network and reduced costs. Finally, we presented two pseudo-polynomial time algorithms based on the conditions developed previously. Although some weakly-polynomial and strongly-polynomial algorithms exist to solve this problem, they require more material. In particular, it should be noted that it is possible to obtain such algorithms with only relatively simple modifications to the one we presented. Finally, as the minimum cost flow is a generalisation of the maximum flow problem, there are further generalisation of the problem like convex flows, generalized flows or multicommodity flows.

References

[1] Ravindra K. Ahuja, . Magnanti, Thomas L., and James B. Orlin. *Network flows : theory, algorithms, and applications / Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin.* Prentice Hall, Englewood Cliffs, N.J. :, 1993.

[2] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms.* McGraw-Hill Higher Education, 2nd edition, 2001.